Ferramentas para a Geração Automática de Documentação

C.O. Braga[1]
A.v. Staa[2]
{arndt,cbraga}@inf.puc-rio.br
Departamento de Informática
Pontifícia Universidade Católica
22453 – 900 Rio de Janeiro,
Brasil

PUC-RioInfMCC 14/95

Junho 1995

Resumo

Este artigo tem por objetivo discutir o problema da geração automatizada da documentação de sistemas de software. São examinadas algumas das tecnologias disponíveis para a solução deste problema. Foram construídas ferramentas para a geração de documentação que poderá ser explorada como um hipertexto, utilizando o sistema de *help* disponível em Windows[3].

Palavras-Chave

Bancos de dados textuais, Documentação, Engenharia de software, Ferramentas, Hipertextos.

Abstract

This paper discusses the automatic generation of software system documents. Some of the available tecnologies are examined. Document generating tools are presented. These tools generate hypertexts which may be explored by means of the Windows help system.

Keywords

Documentation, Hypertext, Software Engineering, Software tools, Textual databases.

---

1 Apoiado pela CAPES, Bolsa de Mestrado, e pela IBM
2 Apoiado pelo CNPq, Bolsa de Pesquisador, e pela Itautec.
3 Windows e Microsoft Help Compiler são marcas registradas da Microsoft.

1. Motivação

O problema de documentar sistemas atinge muitas áreas além da engenharia de software. Por exemplo, não existindo a documentação da rede que passa sob o asfalto, torna-se necessária a realização de uma forma de arqueologia, ou seja abrir buracos no chão para que se descubra por onde passam canos e redes. De forma semelhante, ao lidar com software desenvolvido sem o devido cuidado, torna-se freqüentemente necessário a realização de tarefas de engenharia reversa visando recuperar a especificação e o projeto dos diversos componentes. A manutenção da consistência da especificação do sistema com sua implementação e a viabilização do entendimento rápido e correto da implementação por parte dos mantenedores do software são objetivos a serem alcançados por estas tarefas.

Programas devem ser escritos para que outras pessoas possam lê-los e entendê-los. O enfoque que deve ser dado é o seguinte: "Ao invés de imaginar que nossa tarefa principal é instruir o computador sobre o que ele deve fazer, devemos nos concentrar em explicar a outros seres humanos o que desejamos que o computador faça por nós." É neste enfoque que Knuth [Knuth91] propõe WEB, discutido mais adiante ao falarmos sobre Literate Programming.

A leitura e o entendimento de programas é uma atividade chave no desenvolvimento, na reengenharia de software e na manutenção. O rápido entendimento de um programa está diretamente relacionado com a facilidade de leitura do código e da sua documentação. A utilização de estilo e normas de programação homogêneas aumentam em muito a facilidade com que se faz esta leitura. Aliando-se isto a utilização de ferramentas, tem-se um aumento significativo na qualidade do software e do seu processo de desenvolvimento e manutenção.

O reúso é uma das tônicas modernas no desenvolvimento de software. No entanto, para que se possa reutilizar componentes é necessário o correto e completo registro das especificações e projeto destas componentes.

A complexidade dos sistemas desenvolvidos atualmente deve-se em grande parte a intensa comunicação entre os componentes dos sistemas computacionais. Esta mesma complexidade impõe um desenvolvimento em equipe. Para tornar viável o desenvolvimento em equipe, é necessário que os documentos de cada componente estejam completamente atualizados.

Todas estas observações evidenciam a necessidade de uma boa documentação que seja de fácil e ágil manipulação.

1.1 Browsers e documentação

A documentação de software raramente é lida seqüencialmente. A forma mais comum e intuitiva é a busca de informação especializada, através da exploração do texto utilizando referências cruzadas. Em outras palavras, deve-se apresentar e permitir a exploração da documentação através de um *browser* de hiperdocumento.

Estes hiperdocumentos aparecem nos seguintes formatos:

- documentação para o desenvolvedor do componente. Neste caso, deve-se apresentar a documentação completa e detalhada do elemento;
- documentação para o desenvolvedor que utiliza um dado componente existente. A documentação a ser apresentada neste caso refere-se à parte pública do componente (descrição de uma classe, protótipos das funções, etc.), omitindo detalhes da implementação (informações encapsuladas). Note que este hiperdocumento disponibiliza unicamente a especificação pública do componente. Na documentação do ítem ante-

rior, tanto a especificação encapsulada quanto a pública ficão disponíveis ao desenvolvedor;

- documentação para o usuário da aplicação que contém o componente. Neste caso a documentação deve especificar como operar o componente, sem perder de vista a aplicação como um todo.

Neste artigo apresentamos ferramentas para a geração destes hiperdocumentos.

1.2 A Organização deste artigo

Na seção 2 discutiremos algumas tecnologias disponíveis para a geração automática de documentação. Na seção 3 apresentamos ferramentas alternativas desenvolvidas para gerar hiperdocumentos utilizando o sistema de *help* do Windows. Encerramos o artigo com a proposta de uma ferramenta mais abrangente visando a criação de documentação a partir da análise sintática do código fonte e de marcadores incluídos nos comentários.

2. Tecnologias para solução do problema Documentação

2.1 O Ambiente TALISMAN

A utilização de ferramentas integradas – CASE – é sem sombra de dúvida a melhor forma de resolver este e outros problemas de engenharia de software.

TALISMAN [Staa93] opera sobre repositórios de dados, integrando diferentes representações – forma com a qual se vê o sistema objeto numa determinada fase do ciclo de desenvolvimento de software – através de uma base de software que armazena todos os fatos relativos a um ou mais sistemas. As alterações feitas em uma das representações se propagam automaticamente para as outras representações, eliminando fontes de inconsistência.

O ambiente é adaptável permitindo aos usuários – programadores, analistas, documentadores, gerentes de projeto, inspetores de qualidade – configurar o ambiente às condições particulares do sistema sendo desenvolvido ao estágio do processo de desenvolvimento e às necessidades do próprio usuário. Esta configuração é conseguida através de programas de formulário desenvolvidos pelo usuário de TALISMAN.

Os repositórios de dados operados pelo ambiente são:

- A base de software – armazena informações relativas ao sistema objeto, como diagramas, nomes, fragmentos de texto, fragmentos de código, relacionamentos entre objetos. Pode-se criar programas de formulário que validam, transformam, linearizam, imprimem e exportam as informações contidas no repositório.

- A base de definição – armazena as informações que caracterizam os métodos de trabalho, de forma a adaptar o ambiente às diferentes naturezas dos sistemas com ele desenvolvidos e mantidos. As principais definições contidas na base de definição são:

  ⇒ as linguagens de representação disponíveis – o conjunto de linguagens é adaptável para cada estágio do ciclo de desenvolvimento que estiver sendo trabalhado;

  ⇒ as classes de objeto e seus atributos para cada uma das linguagens;

  ⇒ as definições das linguagens gráficas;

⇒ os programas de formulário de especificação, validação, linearização, exportação, impressão e transformação para cada linguagem;

⇒ mensagens, linguagens de comando, definições de diálogo (parâmetros que controlam o funcionamento de determinada instância do ambiente).

Como foi dito anteriormente, através dos repositórios o ambiente consegue manter consistência entre a especificação e o código implementado. O usuário pode utilizar programas de formulário existentes no ambiente ou então desenvolver os seus próprios, de acordo com a sua necessidade.

Programas de formulário percorrem a base de software produzindo arquivos de texto seqüencial devidamente formatados e acrescidos de títulos e textos constantes. Estes arquivos de texto podem ser submetidos a compiladores para gerar código executável. Através de linearizadores específicos, podem ser gerados arquivos que, quando submetidos à um *help compiler*, geram a documentação que poderá ser explorada como um hipertexto. Programas de formulário são na realidade *queries* ao repositório, permitindo que o usuário obtenha as informações necessárias em um determinado estágio do processo de desenvolvimento ou manutenção do software.

A documentação e o código fonte gerados correspondem exatamente ao que se encontra na base de software, facilitando a edição de especificações e implementações completamente consistentes. A leitura desta documentação é também extremamente fácil, na medida em que o próprio usuário pode organizar a forma de visualizar a informação contida nos repositórios.

2.2 SGML e ODA

Nesta seção apresentamos um outro tipo de tecnologia relacionado ao problema documentação. Tanto SGML (Standard Generalized Markup Language) – [Goldfarb90], [Cowan94] e [Brown89] – como ODA (Office Document Architecture) – [Brown89] e [Appelt93] – são padrões da ISO – ISO 8879 e ISO 8613 respectivamente – para documentos estruturados. A adoção de um padrão facilita o intercâmbio de arquivos entre plataformas. Ambos descrevem uma estrutura estática para os documentos e enfatizam dois aspectos muito importantes: a existência de uma estrutura lógica do documento separada da estrutura de *layout* e a ausência de um modelo padrão para registro e intercâmbio de documentos (portabilidade). A seguir faremos uma breve descrição de cada um destes padrões.

2.2.1 SGML

SGML é uma linguagem para descrição da estrutura de documentos. Dentro do texto são colocados *tags* que identificam os componentes da estrutura do texto.

Um elemento de documento em SGML é definido utilizando-se "<" e ">". Cada elemento começa com <*elem*> e termina com <*/elem*>, onde *elem* é o nome de um componente estrutural. Ilustramos a seguir a redação de um parágrafo:

```
<Parágrafo>
texto te xto textotexto te xtotexto texto
t exto t e x tex to textotexto te xtot exto
</Parágrafo>
```

A estrutura lógica é definida pelo próprio usuário através dos DTDs (Document Type Definition). Um DTD é escrito como se estivéssemos definindo uma nova linguagem de programação utilizando uma meta-linguagem. Um exemplo de DTD segue :

```
<! -- ELEMENT CONTEÚDO -- >
<!ELEMENT DOCUMENT   ( INFOGER, INFOUSO, INFOIMPLEMENT)>

<! -- ELEMENT INFOGER -- >
<!ELEMENT INFOGER     (Projeto, TipoArquivo, LetrasId?, NúmeroId?, Versão, DataAp,
                       Autor+ )>
<!ELEMENT Projeto     ( #PCDATA )>
<!ELEMENT TipoArquivo ( #PCDATA )>
<!ELEMENT LetrasId    ( #PCDATA )>
<!ELEMENT NúmeroId    ( #PCDATA )>
<!ELEMENT Versão      ( #PCDATA )>
<!ELEMENT DataAp      ( #PCDATA )>
<!ELEMENT Autor       ( #PCDATA )>
...
```

Desta forma definimos que o documento é composto por Informações Gerenciais (INFOGER), Informações de Uso (INFOUSO) e Informações de Implementação (INFOIMPLEMENT). Detalhamos um pouco mais as Informações Gerenciais, que seriam compostas por *Projeto* , *TipoArquivo*, *LetrasId* etc., sendo que *LetrasId* e *NúmeroId* são opcionais e *Autor* ocorre 1 ou mais vezes. O que está sendo descrito é, na realidade, a "gramática" do documento.

Em SGML é possível incluir arquivos com outras definições, modularizando a especificação da estrutura lógica do documento. O que realiza a ligação entre a estrutura lógica do documento e sua apresentação são *style-sheets* – regras de formatação definidas junto com a estrutura lógica do documento – que permitem formatar e adequar a apresentação do documento às necessidades do usuário. Pode-se definir um *style-sheet* que, quando aplicado ao documento, produza uma visão *pretty-printed*, e outro que adapte as informações para serem importadas por um determinado compilador ou processador de linguagem [Cowan94].

2.2.2 ODA

O Office Document Architecture é um *framework* orientado a objetos que visa, assim como SGML, padronizar documentos para sua representação e intercâmbio. Apesar de possuírem objetivos comuns, a modelagem do documento em cada uma das propostas é bem diferente.

Como visto na seção anterior, SGML é uma linguagem de marcação (*markup*). ODA é um modelo hierárquico orientado a objetos. Objetos representam componentes do documento e atributos fornecem informação sobre os objetos.

A estrutura lógica do documento é definida pelo formato da árvore e a informação, o texto, fica na folhas desta árvore. Assim como SGML, ODA separa claramente a estrutura lógica do documento da sua apresentação (*lay- out*).

ODA , no entanto, possui uma estrutura de *lay-out* mais bem definida e poderosa do que os *style-sheets* de SGML.

2.2.3 SGML x ODA

As duas propostas tem modelos conceituais diferentes, embora resolvam o mesmo problema. Ambas levantam um ponto importante na criação de documentos que é a existência de uma estrutura lógica independente do *lay-out*.

Uma vez que a estrutura lógica do documento esteja definida, pode-se embutir no código do programa informações relativas a semântica do sistema, uso e gerência e construir *style-sheets* que extraiam estas informações quando necessário.

Entretanto, as duas propostas pecam num ponto crucial quando falamos em documentação de sistemas computacionais: falta suporte à criação de *links* para a formação de hipertextos, ou hiperdocumentos. Estes são a forma mais eficaz de se apresentar documentação de sistemas por facilitar a exploração de documentação. A existência e manutenção da consistência de referências cruzadas dentro dos documentos é de suma importância para o engenheiro de software, tanto na manutenção de sistemas como no desenvolvimento de componentes novas, que necessariamente se conectaram a outras já existentes.

2.3 SGML e bancos de dados textuais

Em [Cowan94] é apresentada uma proposta mais rica do que a que vimos na seção anterior. Usando uma linguagem de marcação, como SGML, a semântica do código pode ser colocada dentro dele e armazenada num banco de dados. Os recursos do banco de dados podem então ser utilizados para se obter diferentes formas de visão do código fonte (*browsing*), permitindo que se explore a documentação através de índices e *hiperlinks*.

2.3.1 O Aspecto Documentação

É evidente que a utilização de bancos de dados textuais enriquece e torna mais poderosa a utilização de linguagens de marcação para a extração de informação do código fonte. A evolução proposta por [Cowan94] junta os conceitos mais importantes verificados neste estudo:

- a existência da estrutura lógica do documento, definida através de um padrão para que o documento seja portátil;
- a utilização de bases de dados textuais, para obtenção das informações inseridas (*embedded*) no código;
- a existência de *hiperlinks* para permitir a navegação, por parte do usuário sobre as visões do código fonte.

Todos estes ingredientes sob a forma de um editor que possa tirar proveito dos recursos do banco de dados textual e disponibilizar as diferentes visões do código e da documentação, permitindo a atualização dos documentos sendo editados através destas visões.

2.4 Literate Programming

Knuth [Knuth91] propõe juntar uma linguagem de formatação de documentos com uma linguagem de programação. Em particular, no seu sistema WEB estão juntas T_EX e Pascal.

O processo de desenvolvimento é o seguinte: o desenvolvedor escreve um arquivo "WEB". No processamento chamado WEAVING é produzido o documento com a descrição do programa de forma natural no formato T_EX, que por sua vez passa pelo compilador T_EX, gerando um arquivo "DVI". O outro processador, TANGLE, produz o código a ser compilado e transformado em código executável. Pelo fato de que o documento descritivo e o código gerado terem a mesma fonte, eles tendem a ser consistentes entre si (da mesma forma que todas as outras tecnologias aqui apresentadas).

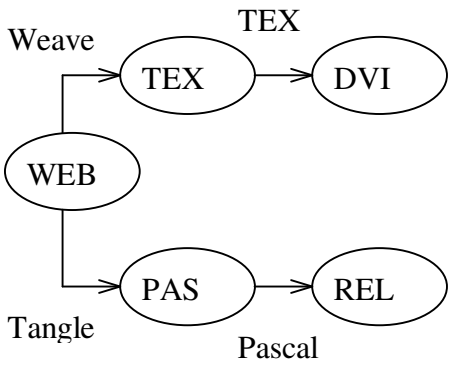

Figura 1 – O uso de WEB

2.4.1 A Documentação gerada por WEB

A proposta de Knuth mostra-se interessante na medida em que propõe uma maneira "desestruturada" de programação, dando liberdade ao programador no processo de criação, o que não ocorre quando utilizando determinadas ferramentas.

T_EX, no entanto, chega a ser mais oneroso ao programador do que SGML, além de não fornecer a definição da estrutura lógica do documento mas somente a formatação do mesmo. Em particular, tem por objetivo produzir documentos de boa qualidade em papel. Infelizmente estes documentos não satisfazem o requisito de facilidade e agilidade de manuseio.

O sistema permite montar referências cruzadas, produzir índices e gerar um documento bem formatado. O mesmo pode ser conseguido com SGML, sem a utilização de bancos de dados textuais. No entanto, não produz um hipertexto.

3. Ferramentas no ambiente TALISMAN

Utilizando TALISMAN, foram construídas duas ferramentas. Estas ferramentas são programas de formulário dinamicamente anexados ao ambiente customizando-o de forma a ter sua funcionalidade ampliada.

3.1 Gerador de documentação de implementação

A documentação técnica de um módulo e de suas componentes é essencial para o desenvolvimento e manutenção de software em equipe. O desenvolvedor e o mantenedor precisam de uma documentação completamente atualizada, que reflita a implementação das componentes a serem utilizadas. Escolhemos o sistema de *help* disponível no Windows como processador de hipertexto. O desenvolvedor cria e mantém especificações e código dos módulos usando TALISMAN. Através de linearizadores especialmente desenvolvidos, geramos arquivos no formato "RTF" que são transformados em *help files* "HLP", por intermédio do Microsoft Help Compiler.

Foram selecionados alguns atributos de especificação de projeto de programa e, a partir daí, construído um hipertexto no formato de *help* Windows. O arquivo *help* é gerado a partir da compilação do arquivo "RTF" produzido por TALISMAN.

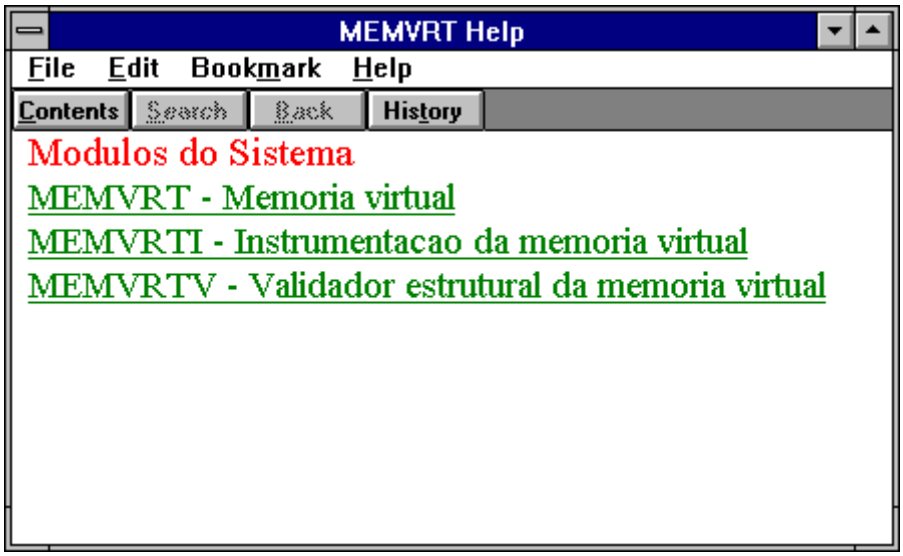

Figura 2 – Tela inicial construída pelo gerador de help

A tabela de conteúdo gerada são *links* para os módulos do sistema. A partir daí pode-se navegar para a descrição mais detalhada de cada módulo e seus componentes, tipos, classes, métodos etc.

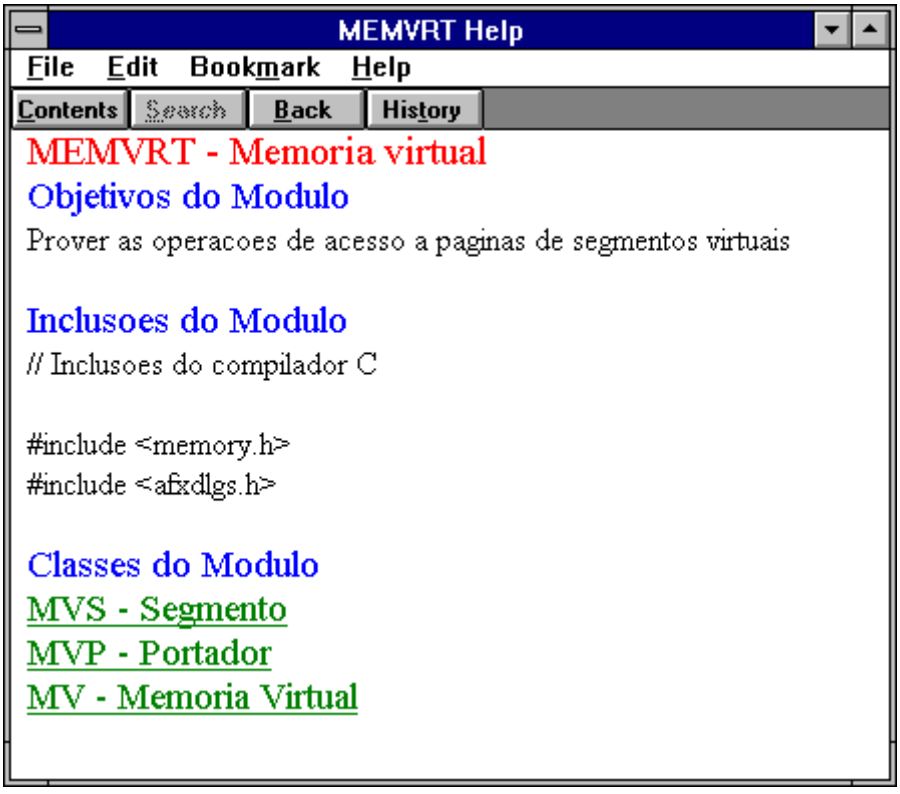

Figura 3 – Parte da descrição de um módulo

A partir dos campos de referências cruzadas existentes na base de software, podemos construir facilmente um hipertexto navegando sobre as informações que desejarmos. Neste protótipo retiramos informações de descrição e relações de composição entre os elementos do projeto modular. Ou seja, existem *links* entre os módulos e suas classes, as classes e os módulos que as utilizam, etc.

Quando se desejar construir o *help* associado a módulos em diferentes BSWs, é necessária a utilização de um programa MAKE que atualiza o arquivo de projeto "HPJ" para que cada arquivo "RTF" gerado seja compilado e ligado para a formação do *help* file final.

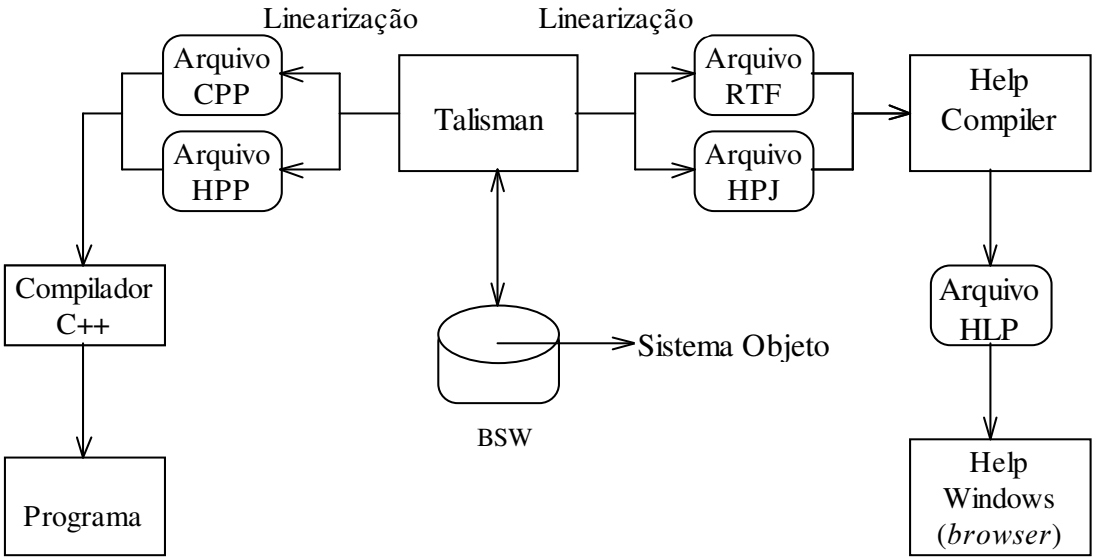

Figura 4 – Processo de geração do arquivo help

A figura 4 descreve o processo de geração de um arquivo *help* a partir da especificação do sistema objeto contida na base de software. Descreve também o processo de geração do código fonte compilável, a partir da mesma base de software. Através do processo de linearização são gerados o arquivo "RTF" e o arquivo de projeto "HPJ" utilizados pelo Microsoft Help Compiler para a geração do arquivo "HLP" em formato Windows.

3.2 Gerador de documentação on-line para o usuário do sistema objeto

A outra ferramenta implementada, também expandindo a funcionalidade do ambiente TALISMAN, automatiza o processo de geração de *help on-line* para o usuário da aplicação desenvolvida no ambiente (sistema objeto).

Como foi mencionado anteriormente, o ambiente é constituído de várias linguagens de representação. Dentre elas estão Estrutura Modular e Documentação.

Na linguagem Documentação é possível construir documentos associados ao sistema objeto. Nela será escrito o arquivo de *help* da aplicação. Para transformar a especificação do documento em um arquivo *help*, foi desenvolvido um outro linearizador para a linguagem Documentação, que produz um arquivo "RTF", que passará pelo processo descrito anteriormente para a geração de arquivos no formato Windows *help File*.

Através da linguagem Estrutura Modular, o programador especifica, projeta e codifica o programa. Com o editor CPP, destinado a especificar, projetar, e implementar programas C++, são definidos os módulos, classes, métodos, tipos e dados. Através do linearizador definido por CPP o código escrito para cada elemento do projeto é transformado no código fonte do programa.

Ao editar os métodos o engenheiro de software pode associar um parágrafo de um documento a este método. Quando o código fonte for gerado, será incorporado a este código a chamada à API do Windows que executa o *browser* de *help* (com o arquivo de *help* linearizado por TALISMAN) associado aquele parágrafo. O processo é mostrado na figura seguinte.

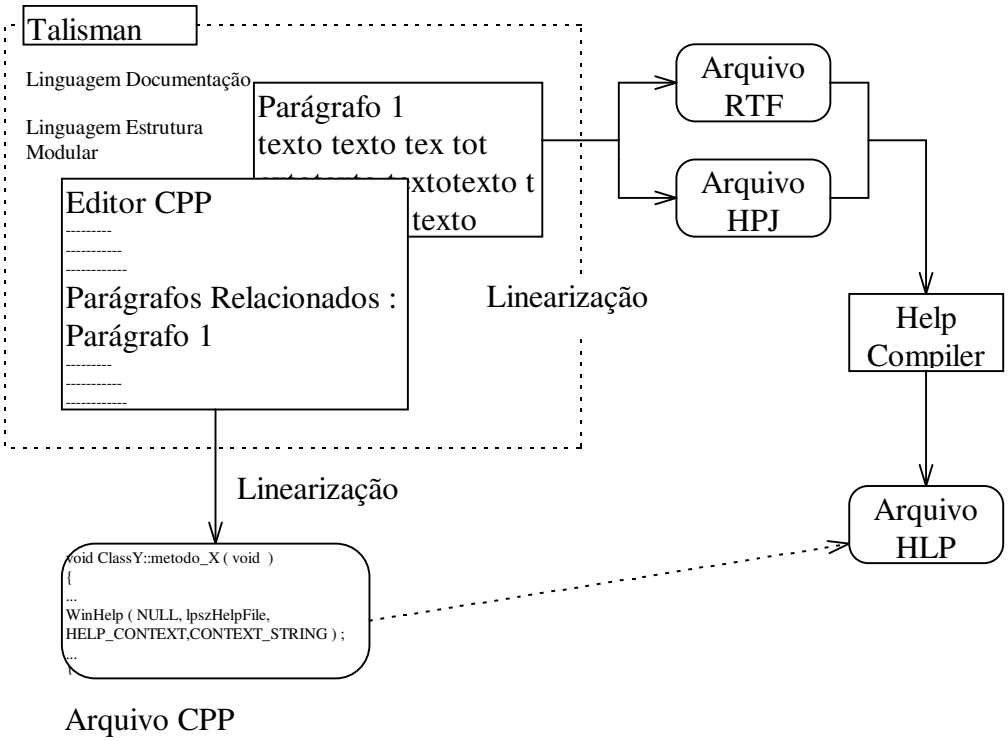

Figura 5 – Processo de geração de help do sistema objeto

Assim como proposto em [Cowan94], TALISMAN se utiliza de um banco de dados para armazenar as informações do sistema objeto. Em relação ao que discutimos anteriormente, dentre as tecnologias expostas, TALISMAN possui uma desvantagem: a forma como os documentos (e também os projetos programa) são construídos, tem uma forma fixa. A estrutura lógica de um documento não pode ser mudada. Um documento é composto por capítulos e estes por parágrafos, que por sua vez referenciam notas de roda pé e/ou figuras. Novos elementos da estrutura lógica de um documento não podem ser definidos como ocorre em SGML (via DTD).

4. Gerador de documentação on-line do sistema objeto

O outro experimento realizado foi construir uma ferramenta que pudesse reconhecer informações embutidas no código fonte sob a forma de comentários, numa linha semelhante à proposta em [Figueiredo94] e [Knuth91]. Estes propõem somente *tags* a serem embutidos nos comentários do código sem a geração de hipertextos.

Em [Staa95] propõe-se um conjunto de regras para a inclusão de especificações em código C/C++. Juntamente a essas regras, definiu-se um conjunto de *tags* incluídos nos comentários para possibilitar a construção de *browsers* destas informações.

O gerador varre o código dos módulos de implementação e através de análise sintática do programa em C++, cria a lista de dependências entre estes módulos. No passo seguinte, extrai as informações contidas nos comentários, segundo os *tags* encontrados. Gera-se assim um hipertexto com estas informações. Note que o analisador da ferramenta tem que ser capaz de identificar todas estas dependências automaticamente e armazená-las num banco de dados como descrito anteriormente.

O hipertexto gerado está no formato Windows *help file*, a partir do processo descrito na seção anterior. O formato “RTF” foi escolhido devido a sua simplicidade. Caso se deseje gerar

documentação em papel, nada impede que se produza um outro linearizador cujo alvo seja T_EX ou os *style-sheets* de SGML.

Um exemplo segue :

```
// – >
//============================================================
//.NOME : <Nome por extenso da classe>
//$CD       Descrição da Classe
//          Texto de descrição
//$CMU Modo de Utilizar a classe
//          Texto descrevendo como utilizar, em particular como utilizar funções virtuais.
//$CED Descrição das Estruturas de Dados
//          No módulo de definição, descreva as assertivas mais abstratas das estruturas
          de
//          dados.
//          No módulo de implementação, defina as assertivas estruturais específicas
// ...
//============================================================
//< –
```

No exemplo acima, definiram-se alguns *tags* para incorporação de especificação de classes no código.

Segue agora uma parte da tabela de *tags* definidos na íntegra em [Staa95].

$ADx	descrição de arquivo. x = C | D | E | I | T C = constantes D = módulo de definição E = módulo de especificação I = módulo de implementação T = dados
$CD	O texto a seguir até o próximo *tag* corresponde a descrição de uma classe
$CMU	O texto a seguir descreve as regras a serem observadas para que se possa utilizar corretamente os métodos da classe
$CED	O texto a seguir descreve as estruturas de dados do ponto de vista do desenvolvedor usuário da classe
$FEx	seção de dados de entrada da função. x = A | E | G | O | P | S A = descrição dos arquivos contendo dados de entrada ou atualização E = descrição dos dados globais encapsulados utilizados antes de serem alterados G = descrição dos dados e tipos globais próprios utilizados antes de serem alterados O = descrição dos dados e tipos globais de outros utilizados antes de serem alterados P = descrição dos parâmetros recebidos S = descrição dos estados da estação requeridos ao entrar
$MDD	descrição do módulo de definição
...	

5. Conclusão e Trabalhos futuros

Com o estudo feito e as ferramentas construídas, concluímos que a geração automática de documentação em forma de hipertexto é um processo viável para a construção correta de software. É viável também a construção de *browsers* para apoio ao desenvolvedor/mantenedor de software e ao usuário do sistema objeto.

A separação entre a estrutura lógica do documento e sua estrutura de *lay-out* (como ocorre em SGML com os DTDs) se mostrou importante, pois podemos construir formas diferentes de visualizar o mesmo documento (*help browsing* e código fonte, por exemplo) adequadas a fins diferentes, porém tendo a mesma estrutura lógica por trás, sendo que esta estrutura lógica se encontra em um padrão que garanta a sua portabilidade. A melhor forma de se

construir o texto a ser utilizado pelo *browser* é através de ferramentas, tais como TALISMAN. Esta ferramenta deve, no entanto, permitir a modelagem da estrutura lógica do documento de forma a permitir que o usuário defina sua própria gramática.

Os experimentos feitos motivaram-nos a iniciar a construção de uma nova ferramenta. Desejamos um editor com as capacidades de *browsing* e formatação descritas em [Cowan94], construindo os *hiperlinks* automaticamente (por análise sintática do código) e de *tags* inseridos no código, armazenando estas informações num banco de dados. É nossa intenção também produzir um editor que siga o proposto em [Brown89], tendo em mente o seguinte :

- manter uma descrição lógica do documento (via SGML);
- permitir a edição desta estrutura através da visão física do documento (permitir a edição dos DTDs) ;
- manter a validade estrutural do documento (regras de validação no DTD);
- conversão entre a forma interna e formas externas permitindo portabilidade (via SGML).

A utilização de SGML como linguagem para estruturação do documento resolve parte das restrições colocadas em [Brown89] para o modelo de editor interativo, inclusive sobre o aspecto portabilidade de documento, que é um dos pontos principais a serem considerados.

Com este trabalho esperamos cobrir todos os aspectos que julgamos importantes no estudo exposto neste artigo, que são :

- a existência de uma estrutura lógica no documento independente do *lay-out*;
- a utilização de um banco de dados para armazenar as informações obtidas da análise do documento estruturado;
- a existência de uma interface interativa que permita ao usuário alterar o documento nas diferentes formas de visão disponíveis, sem que a estrutura do documento seja modificada;
- a criação automática de *hiperlinks* pela análise do documento e consequentemente a disponibilização de um hipertexto para o usuário;
- a existência de mecanismos de validação da estrutura do documento e dos *hiperlinks*.

6. Referências bibliográficas

[Appelt93] Appelt, W.; Tetteh-Lartey, N.; "The Formal Specification of the ISO Open Document Architecture (ODA) Standard"; *The Computer Journal*; Vol. 36; No. 3; 1993

[Brown89] Brown, H.; "Standards for Structured Documents"; *The Computer Journal*; Vol 32, No. 6 ; 1989; páginas 505-514

[Cowan94] Cowan, D.D.; Germán, D.M.; Lucena, C.J.P.; Staa, A.v.; "Enhancing Code for Readability and Comprehension using SGML"; *Computer Science Department*; University of Waterloo; ONT; Canadá; março;1994

[Figueiredo94] Figueiredo, L.H.; Filho, W.C.; "Documentação de Software no TecGraf"; Relatório técnico do grupo de tecnologia em Computação Gráfica; Departamento de Informática; PUC – RIO; Rio de Janeiro; Brasil

[Goldfarb90] Goldfarb, C.; *SGML Handbook*; Oxford University Press, 1990

[Knuth91] Knuth, D.E.; *Literate Programming*; Center for the Study of Language and Information; U.S.; 1991

[Staa93] Staa, A.v.; *Ambiente de Engenharia de Software TALISMAN, Manual do Usuário*; Rio de Janeiro; Staa Informática; 1993

[Staa94] Staa, A.v.; Cowan, D.D.; "An Overview of the Totem Software Engineering Meta – Environment"; *Computer Science Department*; University of Waterloo; ONT, Canadá; setembro,1994

[Staa95] Staa, A.v.; Costa, G.M.; "Regras e recomendações para a inclusão de especificações no código de programas C ou C++"; *Monografias em Computação*, Departamento de Informática, PUC-Rio (em desenvolvimento)